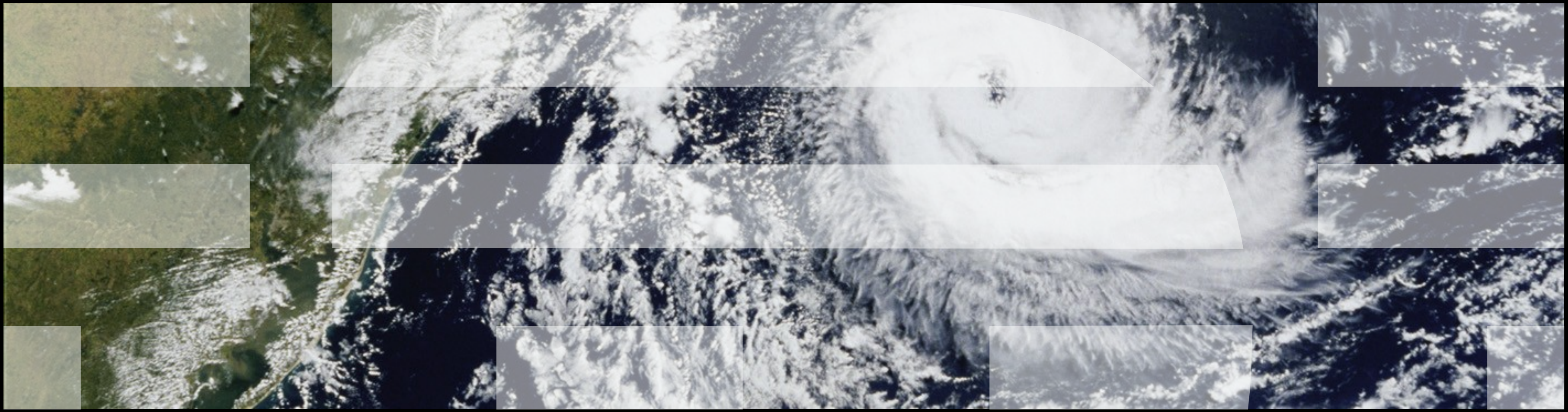
Paul E. McKenney, IBM Distinguished Engineer, Linux Technology Center

2012 RTLWS Chapel Hill, NC USA October 19, 2012

On-Chip Cache Coherence and Real-Time Systems

And What is New in RCU for Real Time



Overview

- On-chip cache coherence and real-time systems
- New real-time features for RCU
- Other future RCU work

On-Chip Cache Coherence and Real-Time Systems

On-Chip Cache Coherence and Real-Time Systems

- July 2012 CACM: “Why on-chip coherence is here to stay”, Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin
  - Argued that for real-fast systems, cache-coherence will persist indefinitely
    - Cache coherence: all CPUs agree on the data in a given cache line
    - No need for cache-flush instructions (just the usual memory barriers)

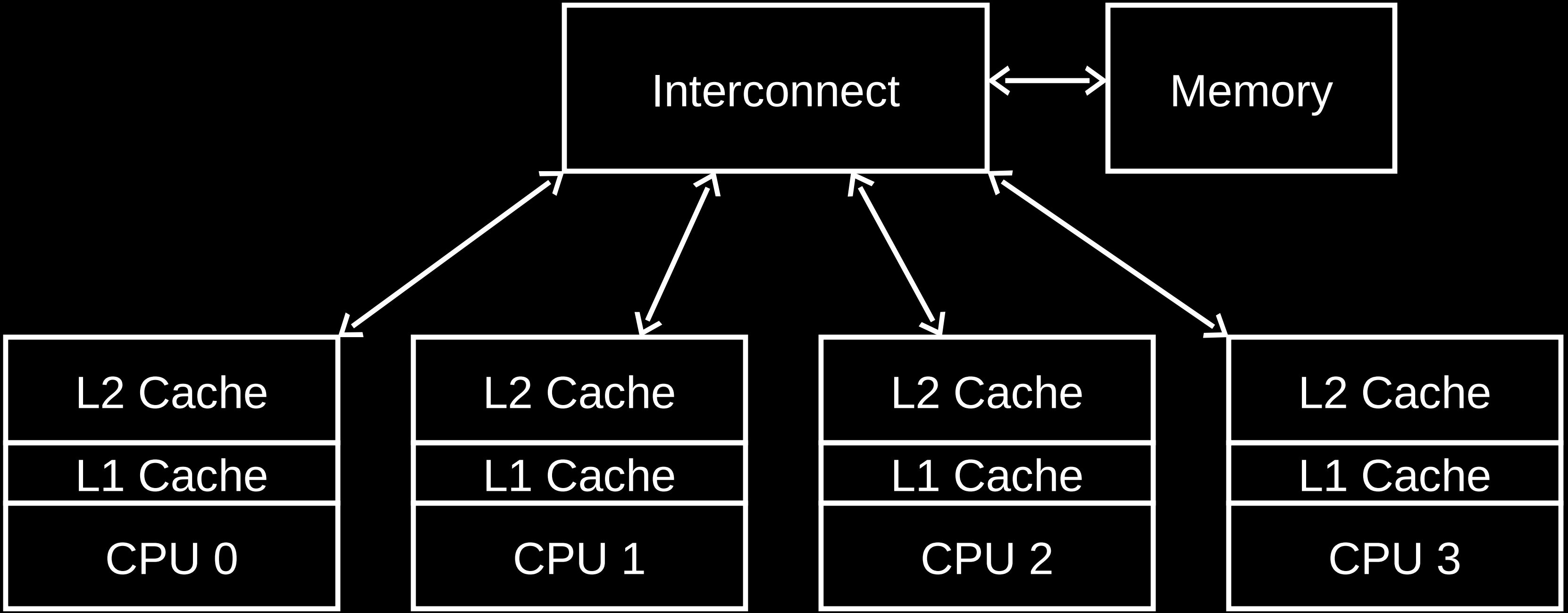
On-Chip Cache Coherence and Real-Time Systems

- July 2012 CACM: “Why on-chip coherence is here to stay”, Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin
  - Argued that for real-fast systems, cache-coherence will persist indefinitely
    - Cache coherence: all CPUs agree on the data in a given cache line
    - No need for cache-flush instructions (just the usual memory barriers)
  - Which should be a relief to us software guys writing parallel code
    - After all, memory barriers cause enough trouble, don't they?

On-Chip Cache Coherence and Real-Time Systems

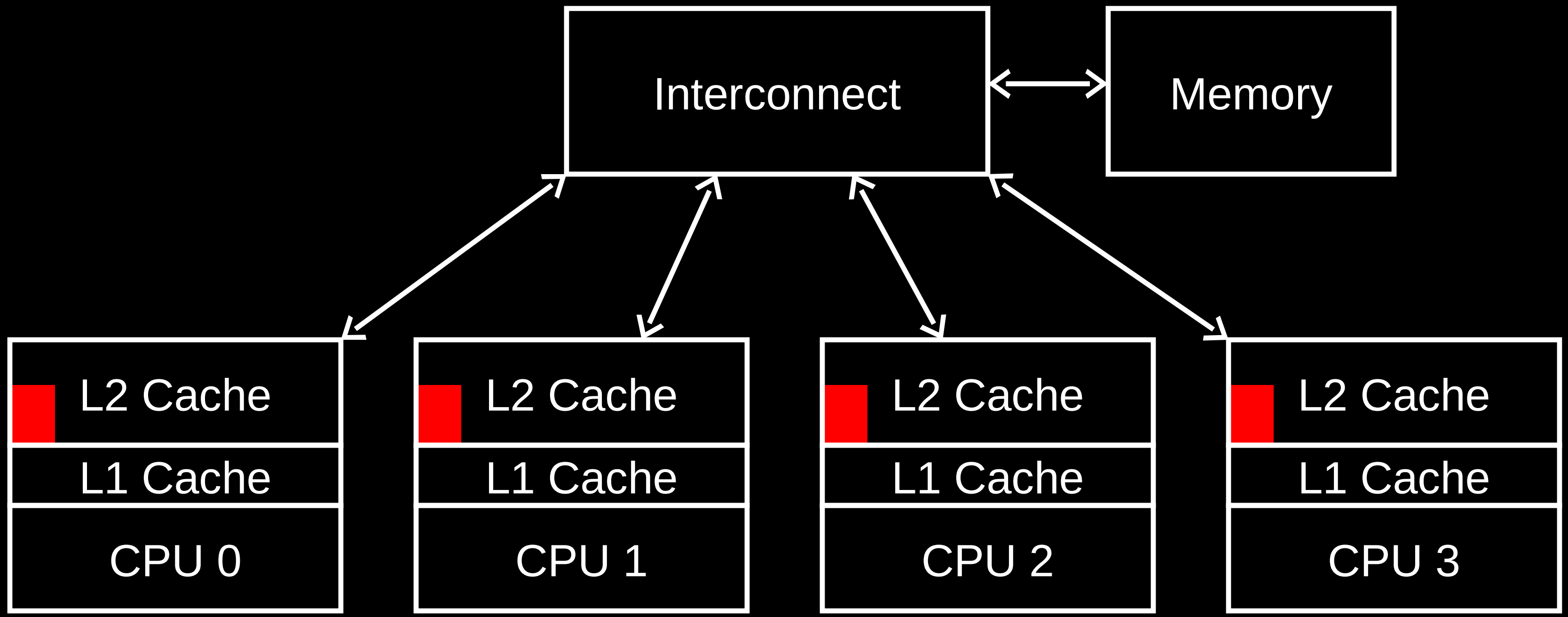
- July 2012 CACM: “Why on-chip coherence is here to stay”, Milo M. K. Martin, Mark D. Hill, and Daniel J. Sorin
  - Argued that for real-fast systems, cache-coherence will persist indefinitely
    - Cache coherence: all CPUs agree on the data in a given cache line
    - No need for cache-flush instructions (just the usual memory barriers)
  - Which should be a relief to us software guys writing parallel code
    - After all, memory barriers cause enough trouble, don't they?
  - But what about real-time systems?

How Cache Coherence Is Implemented in Hardware



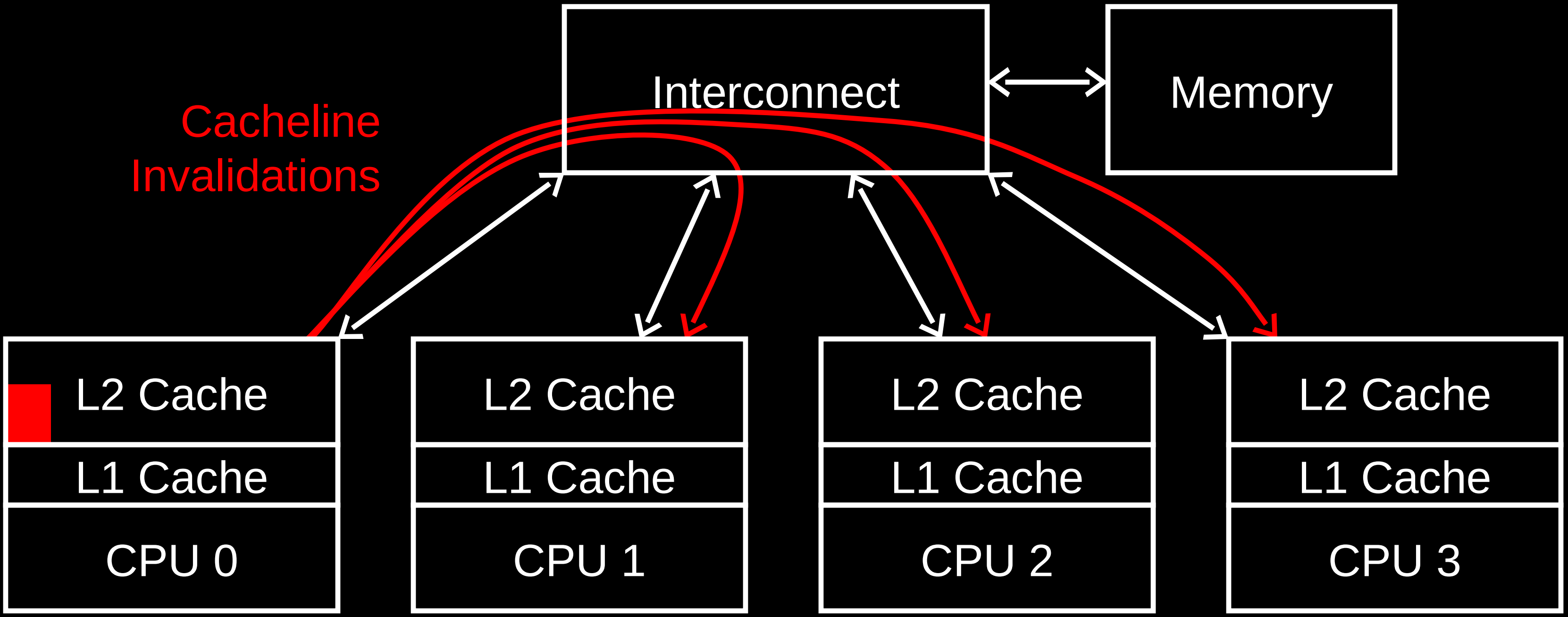
How Cache Coherence Is Implemented in Hardware

Shared Variable in Red
All Reads Local (Fast)



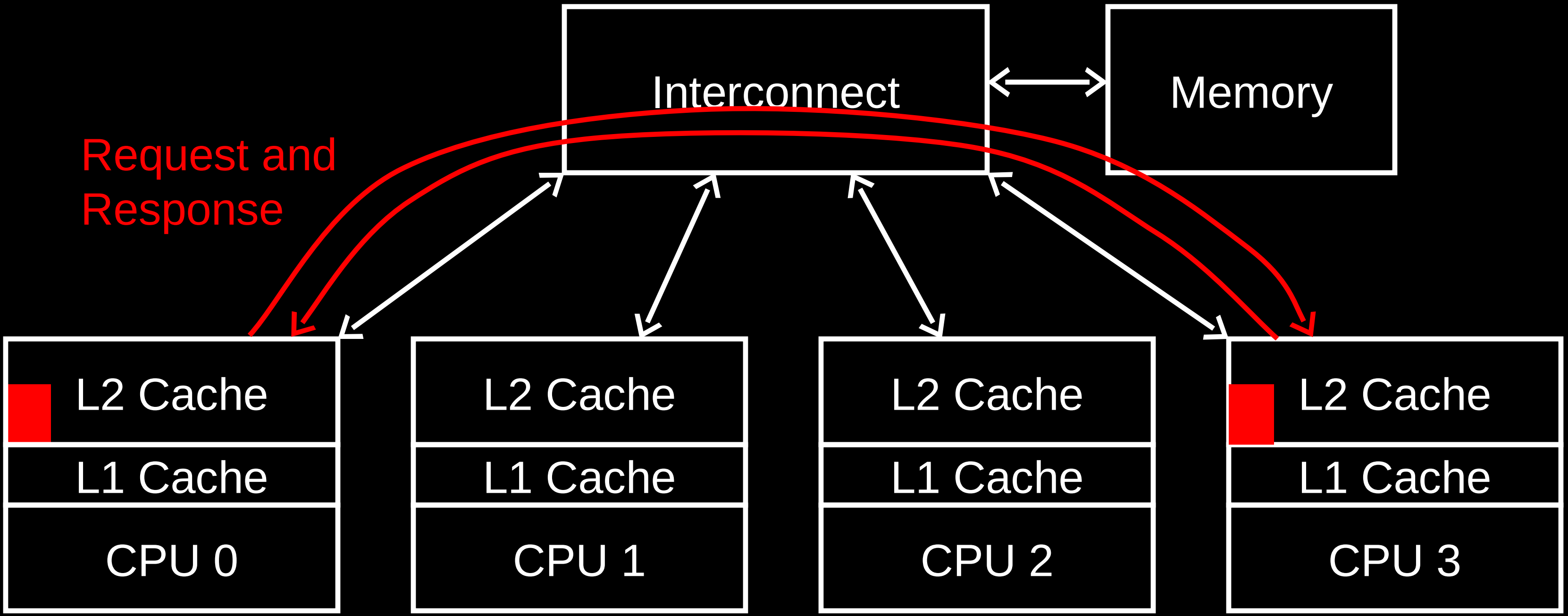
How Cache Coherence Is Implemented in Hardware

Shared Variable in Red:
CPU 0 Updates it (Slow)



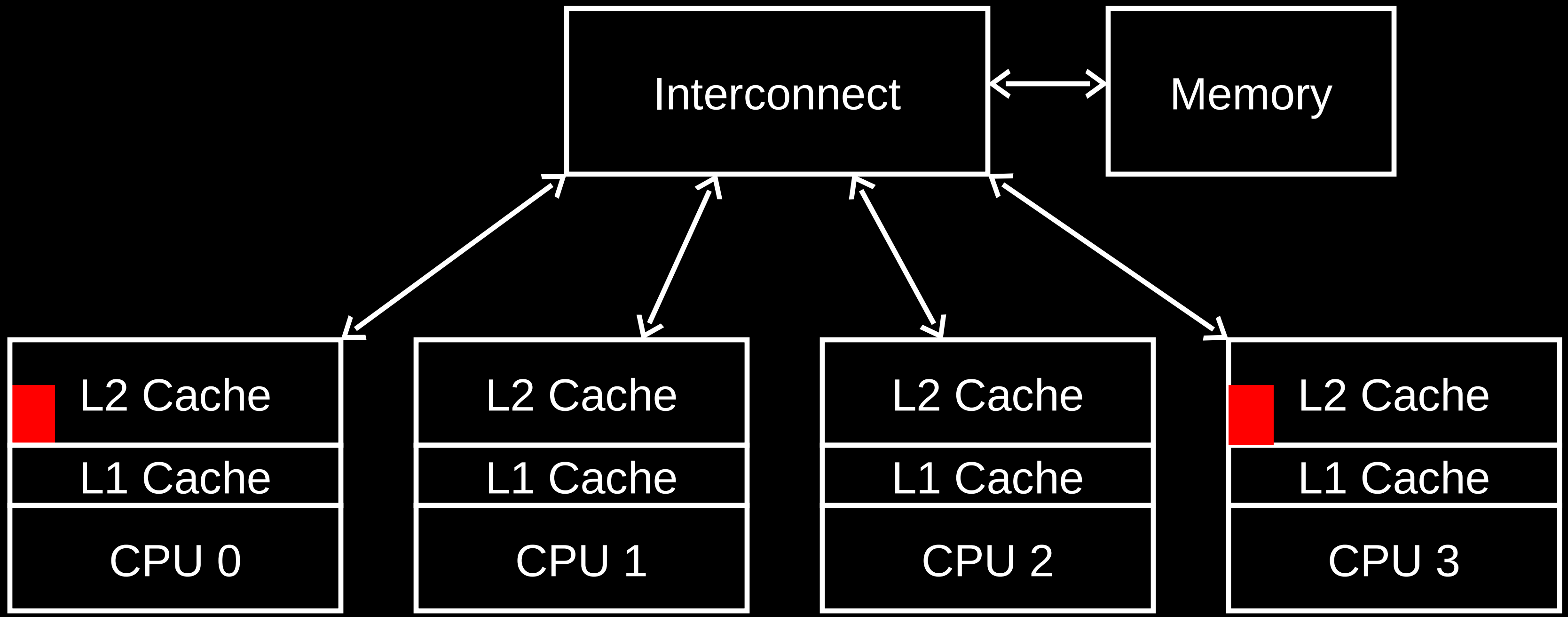
How Cache Coherence Is Implemented in Hardware

Shared Variable in Red:
CPU 3 Reads (Slow)



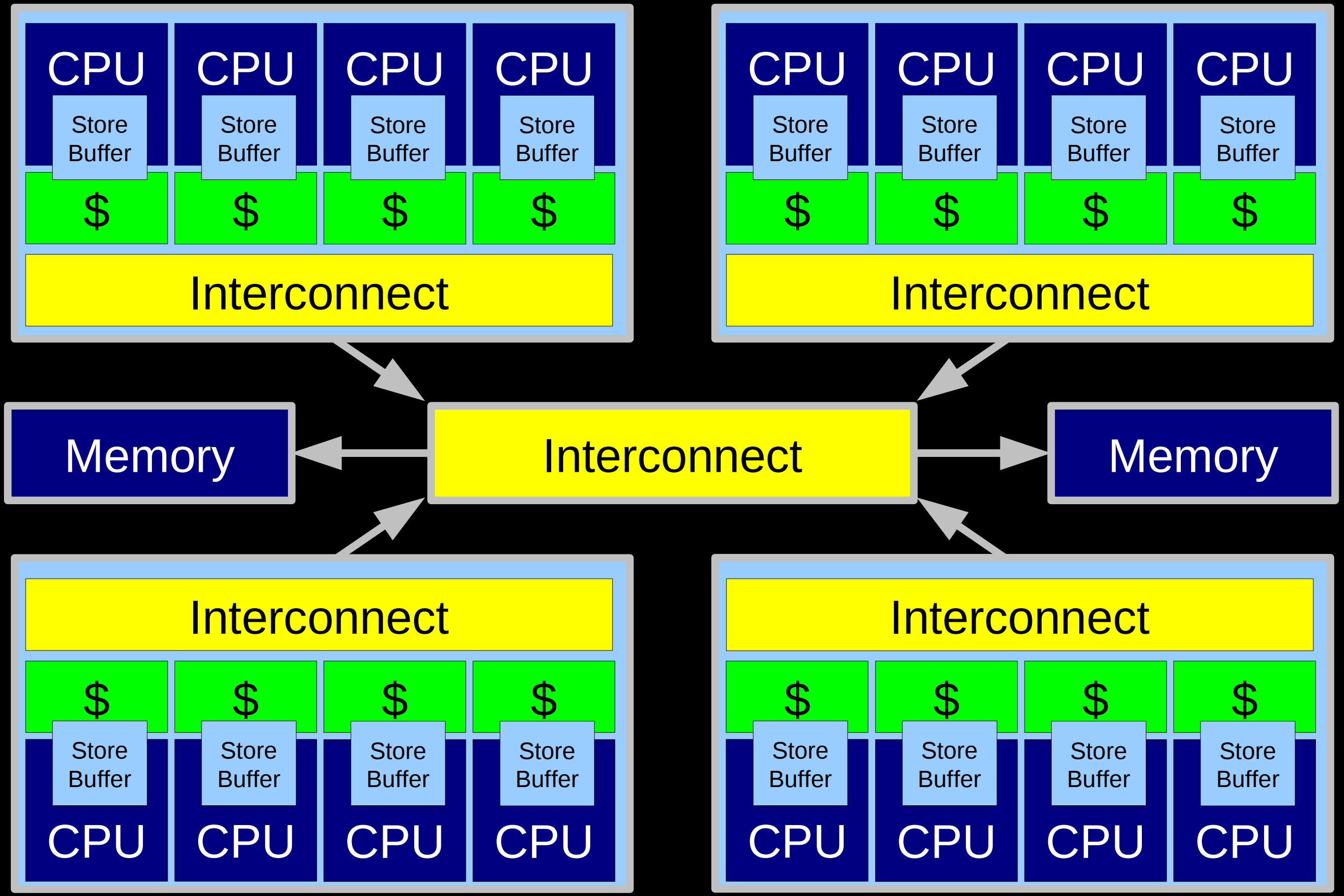
How Cache Coherence Is Implemented in Hardware

Shared Variable in Red:
CPUs 0 and 3 Reads (Fast)



Lots of Effort Required From Hardware! And Today's Systems Have More CPUs...

Modern Multicore System Architecture



More work for hardware to maintain cache coherence

Will Systems Continue to be Cache Coherent?

Modern Multicore System: Cache-Coherence Issues

- Broadcasting invalidations could result in $O(N^2)$ traffic
  - Directory-based cache-coherence schemes send messages only where needed – but that could still be a lot of traffic, N^2 worst case!
- Directory-based cache-coherence schemes add hardware
  - Minimize added hardware via “inclusion”: any line in a cache close to a CPU is also maintained by all levels farther from that CPU
  - Further reduced by increasing number of levels in cache hierarchy
- Maintaining inclusion can result in needless rollouts
  - Can eliminate these by increasing associativity: shared cache associativity must equal sum of subordinate caches
    - Usually infeasible: 8x 8-way caches means 64-way shared cache!
  - But decreasing associativity to 16 ways results in small miss rate
- Taller cache hierarchy means more memory latency
- Energy efficiency???

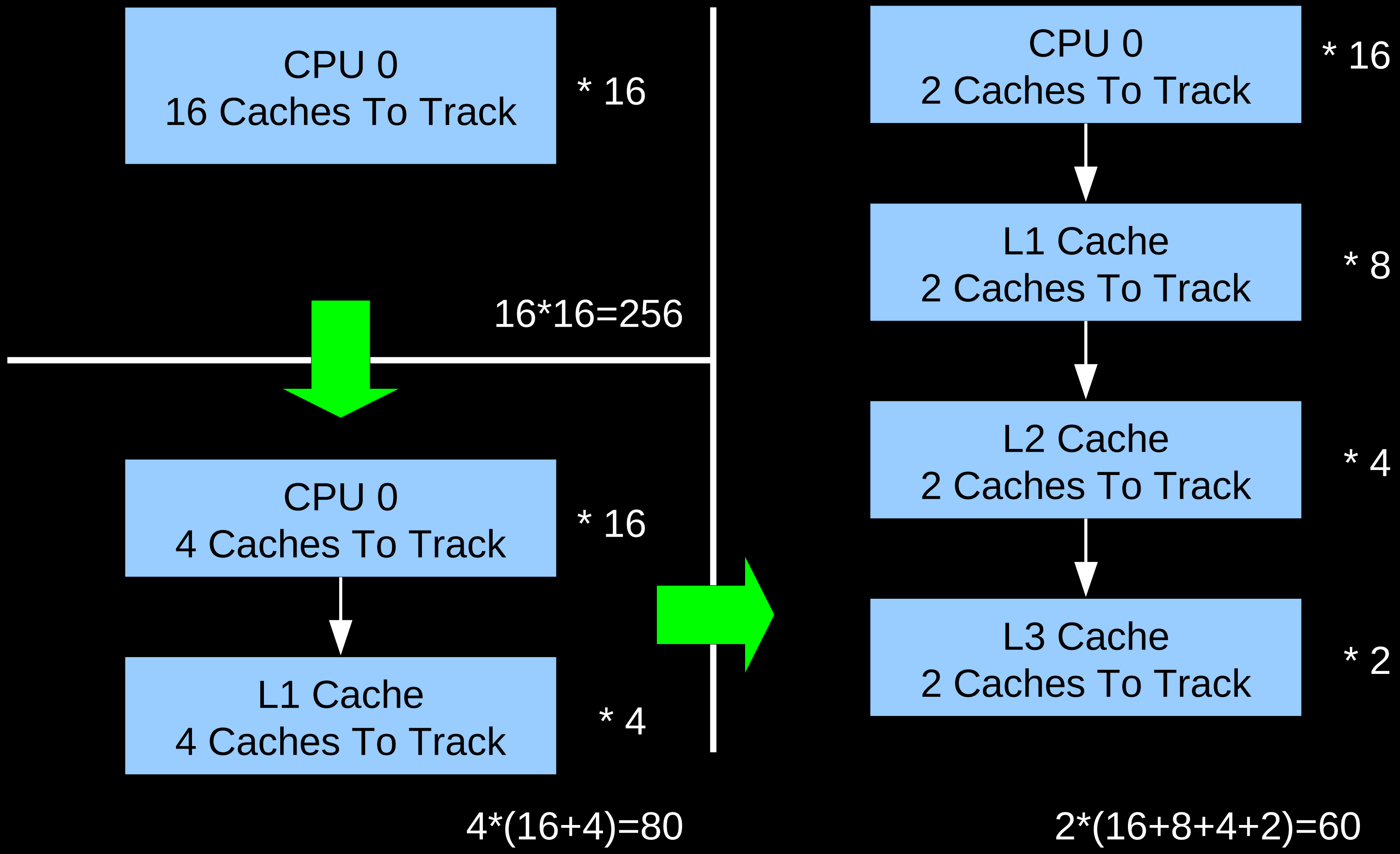
Broadcasting Invalidations and N^2 Traffic

- Worst-case invalidation traffic is still $O(N^2)$
  - –And the worst case is what real-time is all about...
- For real-fast systems, this is not a problem:
  - –Directory-based system: Invalidations only sent where needed
  - –Every cache holding the cache line got it via a cache miss
  - –Hardware can process invalidations in parallel
  - –Average per-access invalidation overhead thus sharply bounded
- This doesn't help for real-fast systems: What to do?
  - –Measure worst-case invalidation
  - –If too large, use software techniques to limit sharing
    - Partitioning, hierarchy, …
    - For extra credit, adjust the jiffies counter for real-time usage...

Directory-Based Cache Coherence

- Directory-based cache-coherence mostly invisible to real-time
  - Except for cache-miss and cache-level effects due to need for inclusion
- For hardest real time, you pretty much need to assume all accesses miss the cache
  - But most real-time systems are not quite that hard
  - And are probably just stuck with the added latency, work around it by:
    - Using a fraction of the CPUs, based on cache size (similar to turning off hyperthreading)
    - Engineer a safety factor to allow for increased cache-miss rate
    - Use special CPUs designed for real-time embedded work
- Taller cache hierarchy means more memory latency
  - Hopefully increases in cache size help to counteract this trend, at least for softer forms of real-time systems
  - Real-fast costs for too-tall cache hierarchy will limit the real-time pain
    - See next slide
  - Special real-time embedded CPUs might still be needed

Effects of Adding Cache Levels For 16-CPU System



CPU 0
16 Caches To Track
* 16

16*16=256

CPU 0
4 Caches To Track
* 16

L1 Cache
4 Caches To Track
* 4

4*(16+4)=80

CPU 0
2 Caches To Track
* 16

L1 Cache
2 Caches To Track
* 8

L2 Cache
2 Caches To Track
* 4

L3 Cache
2 Caches To Track
* 2

2*(16+8+4+2)=60

Energy Efficiency???

- Real-time systems still seem to turn off energy-efficiency features
- But it is likely that continued energy-efficiency progress will come at the expense of real-time response
- And sooner or later, there will be a demand for energy-efficiency real-time systems
- Thomas Gleixner says: “If your deadlines allow enough time to power things up, there is no reason not to combine energy efficiency with real-time response”
  - Though people can be expected to want to push the envelope on both energy efficiency and real-time response
  - Which might be another good reason for a deadline scheduler

Energy Efficiency???

- Real-time systems still seem to turn off energy-efficiency features
- But it is likely that continued energy-efficiency progress will come at the expense of real-time response
- And sooner or later, there will be a demand for energy-efficiency real-time systems
- Thomas Gleixner says: “If your deadlines allow enough time to power things up, there is no reason not to combine energy efficiency with real-time response”
  - Though people can be expected to want to push the envelope on both energy efficiency and real-time response
  - Which might be another good reason for a deadline scheduler
- Boredom is still not a short-term problem!

RCU and Real Time: History and Progress

What The Heck Is RCU???

- For an overview, see http://lwn.net/Articles/262464/
- For the purposes of this presentation, think of RCU as something that defers work, with one work item per callback
  - Each callback has a function pointer and an argument
  - Callbacks are queued on per-CPU lists, invoked after grace period
    - Invocation can result in OS jitter and real-time latency
  - Global list handles callbacks from offlined CPUs: adopted quickly
- And that has read-side critical sections
- And that is a state machine driven out of scheduler_tick(), softirq, and kthread(s)

RCU and Real Time: History

- 2005: Preemptible RCU take 1 (in -rt)
- 2007: Preemptible RCU take 2: nonatomic (in mainline)
- 2009: Preemptible RCU take 3: scalable (in mainline)
- 2012: Bug report claiming 200-microsecond latency spikes from RCU grace-period initialization

RCU and Real Time: History

- 2005: Preemptible RCU take 1 (in -rt)
- 2007: Preemptible RCU take 2: nonatomic (in mainline)
- 2009: Preemptible RCU take 3: scalable (in mainline)
- 2012: Bug report claiming 200-microsecond latency spikes from RCU grace-period initialization
  - Which came as quite a surprise given ~30-microsecond latencies from the entire kernel, not just RCU...

RCU and Real Time: History

- 2005: Preemptible RCU take 1 (in -rt)
- 2007: Preemptible RCU take 2: nonatomic (in mainline)
- 2009: Preemptible RCU take 3: scalable (in mainline)
- 2012: Bug report claiming 200-microsecond latency spikes from RCU grace-period initialization
  - Which came as quite a surprise given ~30-microsecond latencies from the entire kernel, not just RCU...
  - But further down in the email, there was a kernel-configuration parameter that fully explained the difference in latency

RCU and Real Time: History

- 2005: Preemptible RCU take 1 (in -rt)
- 2007: Preemptible RCU take 2: nonatomic (in mainline)
- 2009: Preemptible RCU take 3: scalable (in mainline)
- 2012: Bug report claiming 200-microsecond latency spikes from RCU grace-period initialization
  - Which came as quite a surprise given ~30-microsecond latencies from the entire kernel, not just RCU...
  - But further down in the email, there was a kernel-configuration parameter that fully explained the difference in latency
  - NR_CPUS=4096!!!
    - At which point: “You mean it *only* took 200 microseconds???”
    - Therefore...

RCU and Real Time: History

- 2005: Preemptible RCU take 1 (in -rt)
- 2007: Preemptible RCU take 2: nonatomic (in mainline)
- 2009: Preemptible RCU take 3: scalable (in mainline)
- 2012: Preemptible grace-period handling (in mainline)
  - Who knew that 4096-CPU systems would do real-time work???

RCU and Real Time: History

- 2005: Preemptible RCU take 1 (in -rt)
- 2007: Preemptible RCU take 2: nonatomic (in mainline)
- 2009: Preemptible RCU take 3: scalable (in mainline)
- 2012: Preemptible grace-period handling (in mainline)
  - Who knew that 4096-CPU systems would do real-time work???
  - Of course, limited 4096-CPU testing implies likely remaining bugs...
  - And still need debugging features such as tracing

RCU and Real Time: Ongoing Work

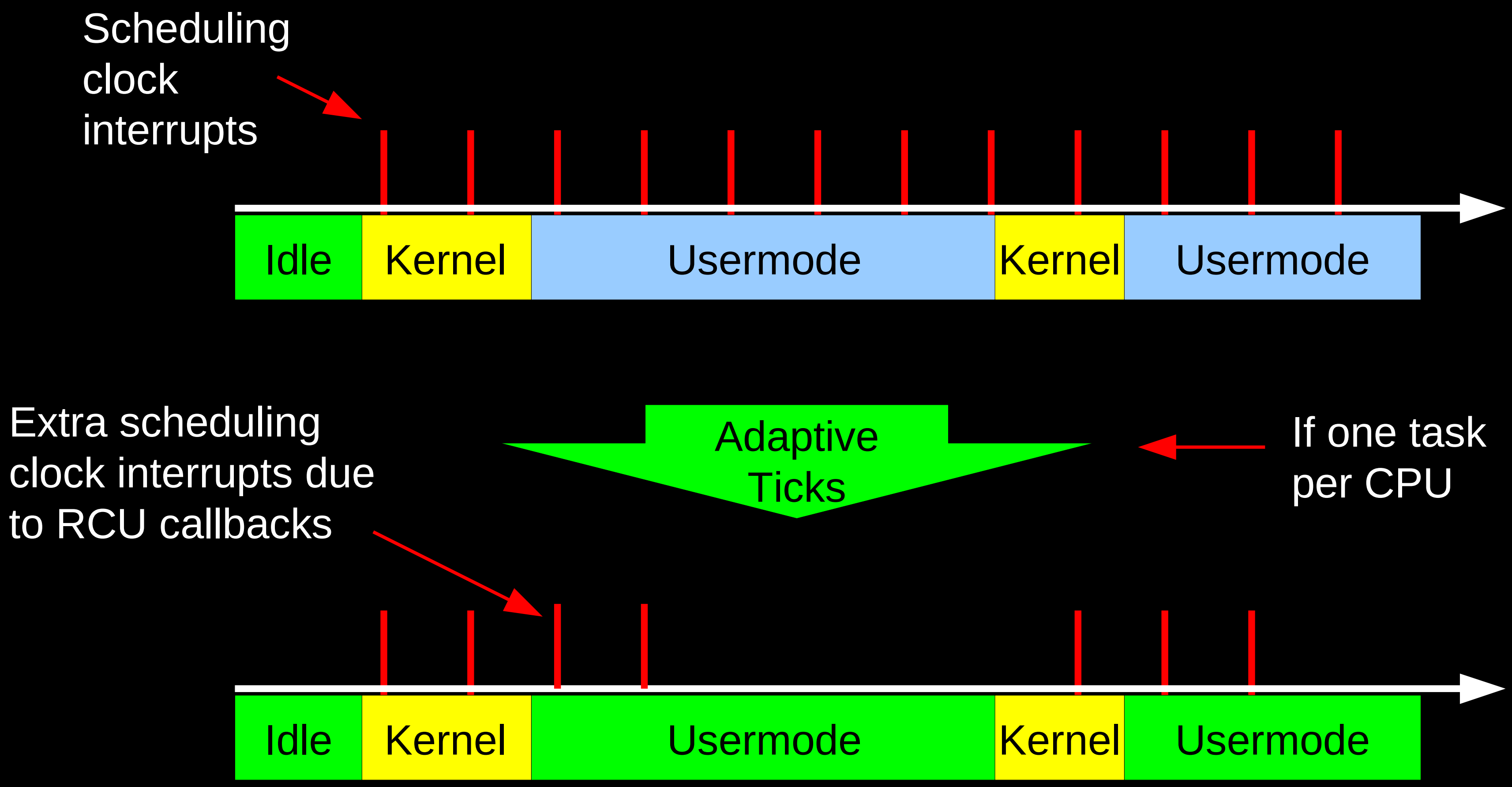
RCU and Real Time: Ongoing Work

- 2011-: Preparation for Frederic's adaptive ticks (in mainline):
  - Lots of dyntick-idle work preparing for adaptive ticks
    - Less OS jitter for usermode execution once complete
  - rcu_barrier() done, synchronize_sched_expedited() and synchronize_rcu_expedited() still need additional work
- 2011-: “Lazy” RCU callbacks
  - Lai Jiangshan Introduced kfree_rcu(), need other variants
- 2012-: Offloading callbacks from selected CPUs
  - Initial report at Linux Plumbers Conference
  - Embarrassingly little progress since then
- 2012-: Get rid of RCU-bh once uses are removed
  - Reduce -rt diffs for RCU

Preparation For Adaptive Ticks

- RCU modifications to support Frederic's adaptive ticks
- RCU treats user-mode execution as idle, reducing the need for scheduling-clock interrupts in user-mode execution
  - Thereby reducing OS jitter and improving real-time response
  - Also removing rcu_barrier() interruptions
    - And, later, interruptions from _expedited primitives
- Still have RCU disturbance due to CPUs having RCU callbacks queued when transitioning to usermode execution

Preparation For Adaptive Ticks: Graphical View

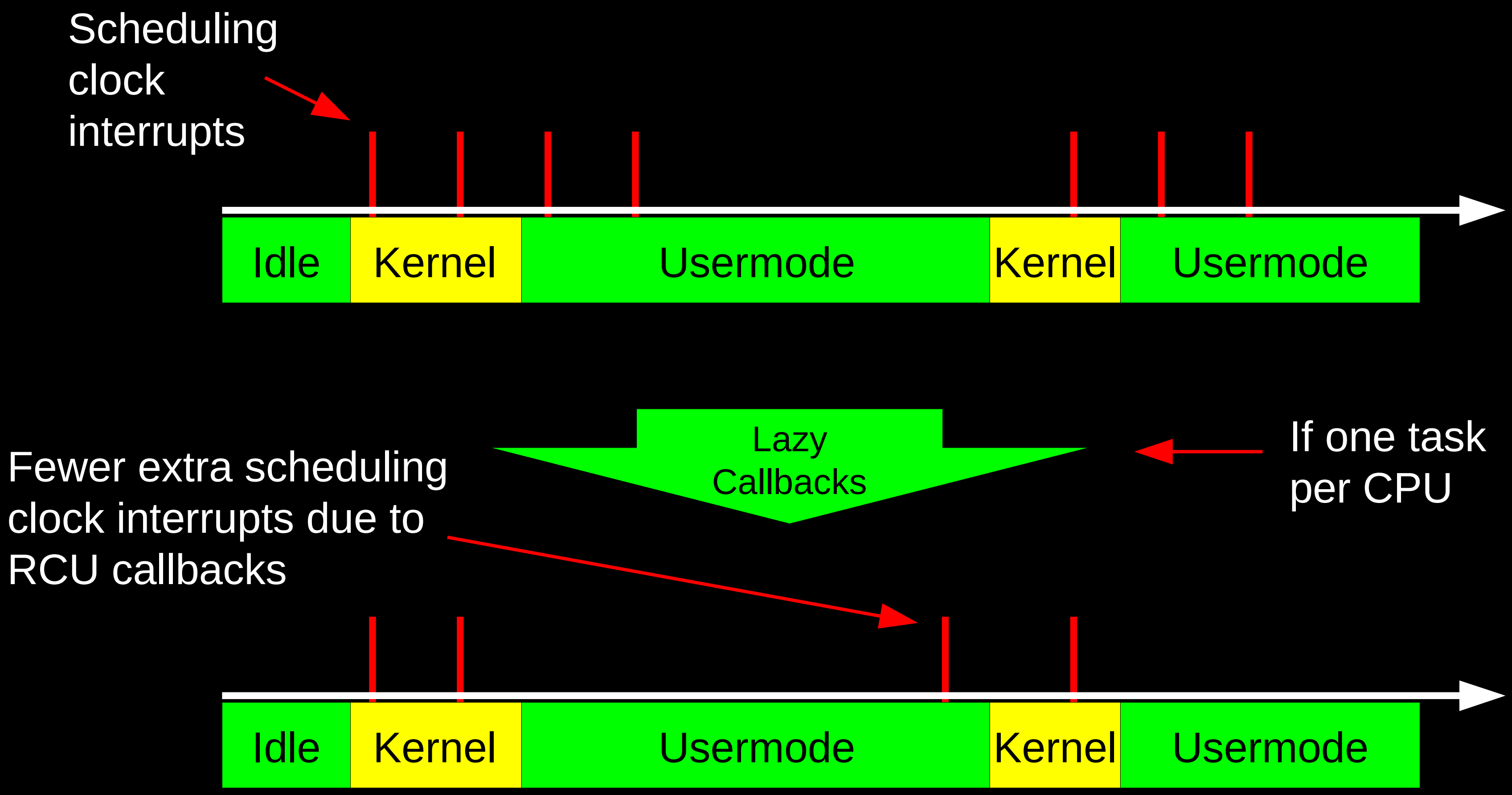


Reduce OS jitter for real-time and HPC workloads

“Lazy” RCU Callbacks

- Some RCU callbacks wake threads up
  - Thus need to be processed in a timely fashion
  - Indefinite postponement might well mean a system hang
- Other RCU callbacks only free memory
  - As long as the system has ample memory, can defer indefinitely
    - For values of “indefinitely” equal to ten seconds
  - Thus reducing OS jitter and improving energy efficiency
- Lai Jiangshan Introduced kfree_rcu() for this purpose
  - But this does not handle deferred free to slabs
- Very likely also need call_rcu_lazy()
- However, all of this is low priority
  - High dynamic proportion of callbacks do non-trivial work

“Lazy” RCU Callbacks: Graphical View



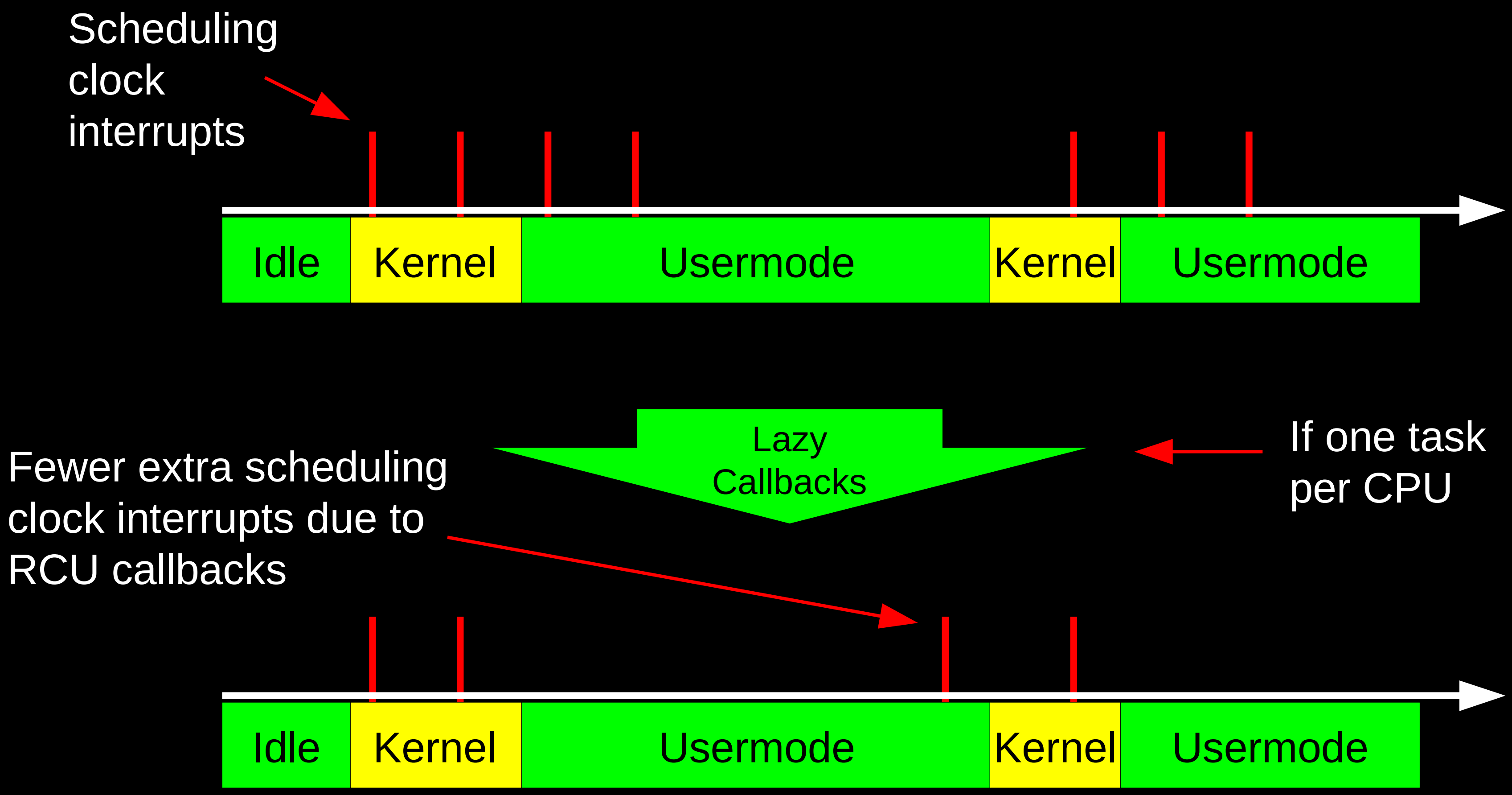
Scheduling clock interrupts

If one task per CPU

Fewer extra scheduling clock interrupts due to RCU callbacks

But what if you want *no* scheduling clock interrupts to userspace applications?

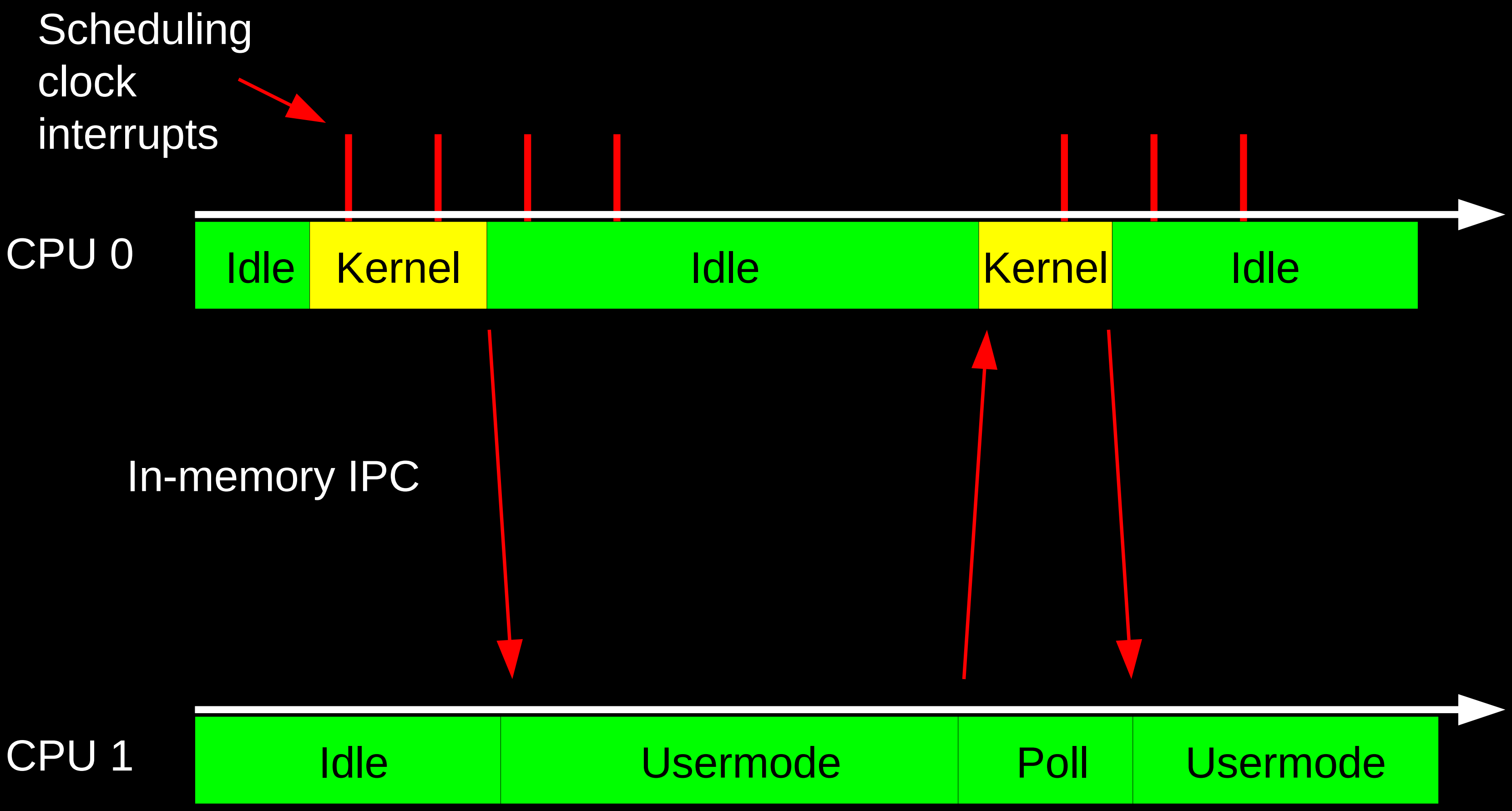
“Lazy” RCU Callbacks: Graphical View



But what if you want *no* scheduling clock interrupts to userspace applications?

Don't do interrupts or system calls on that CPU!!!

No System Calls or Interrupts: Graphical View



Don't do interrupts or system calls on that CPU,
so extra scheduling clock interrupts due to RCU callbacks!!!

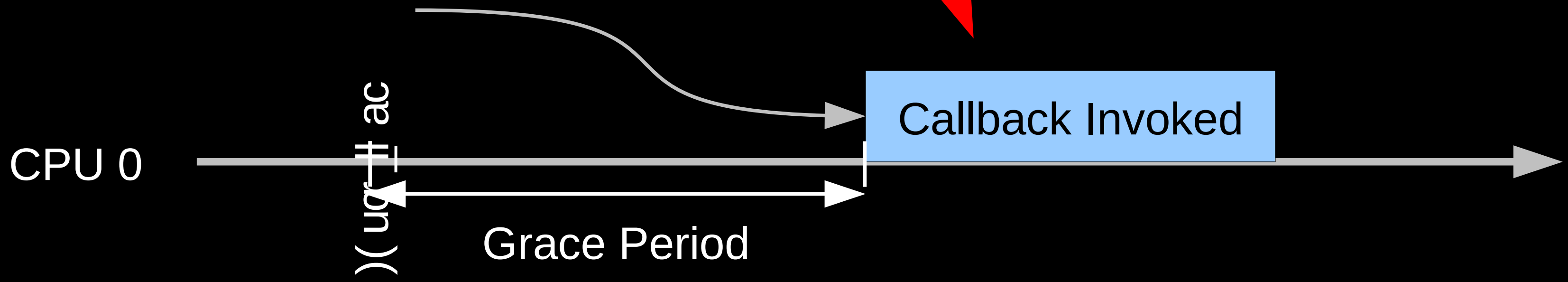
But Sometimes You Really Need On-CPU Syscalls...

But Sometimes You Really Need On-CPU Syscalls...
So Offload the RCU Callbacks!

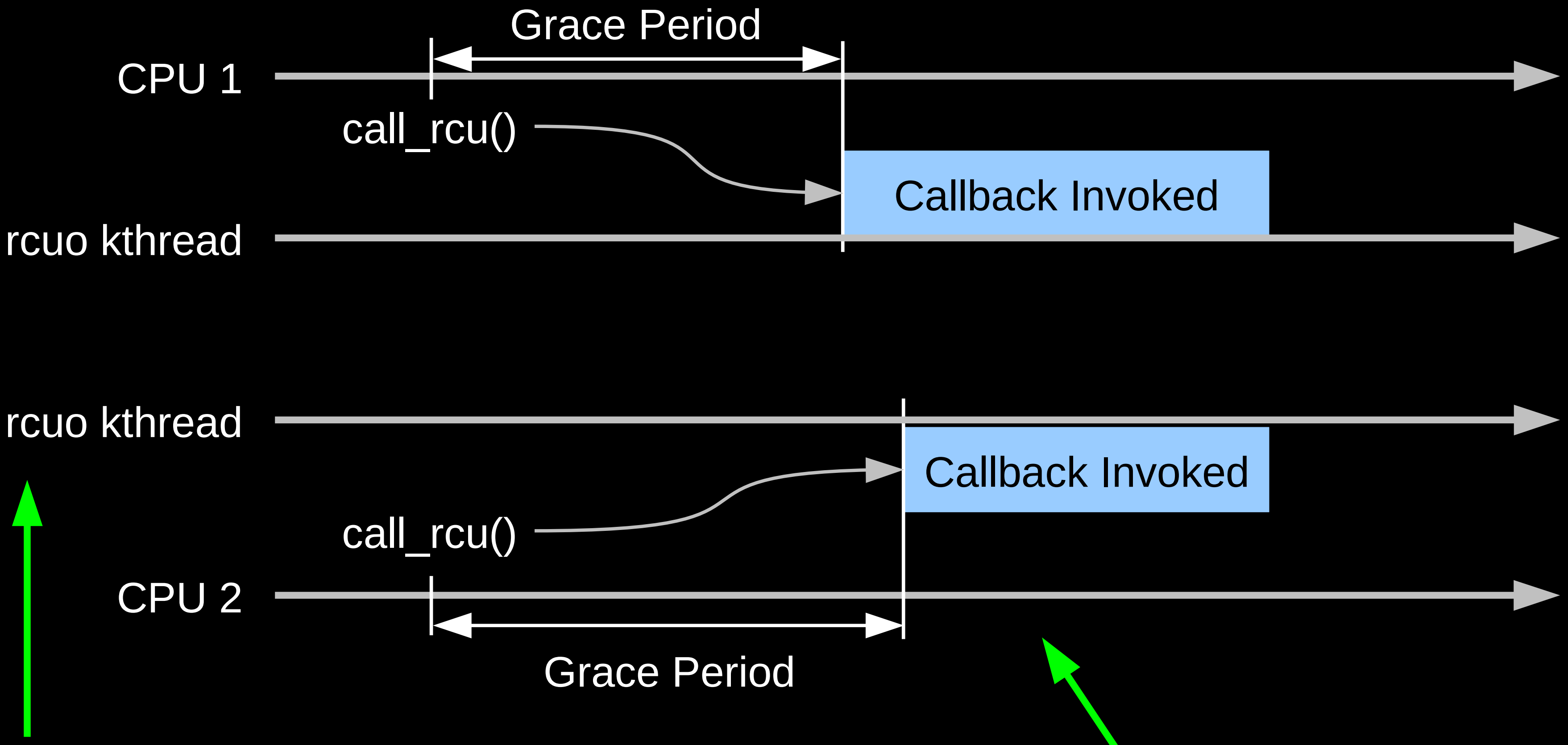
Offloading RCU Callbacks From Selected CPUs

- The problem with RCU callbacks:

Likely disrupting whatever was intended to execute at about this time...



RCU Callbacks, Houston/Korty for TREE_RCU



Scheduler controls placement
(or can place manually)

No disruption!

Offloadable RCU Callbacks: Limitations and Futures

- Must reboot to reconfigure no-CBs CPUs
  - rcu_nocb_poll kernel command-line parameter gives list of no-CB CPUs
  - Races between reconfiguring, registering callbacks, rcu_barrier(), grace periods and who knows what all else are far from pretty! (But you can move the kthreads around w/out boot.)
- Scalability: 1,000 no-CBs CPUs would not do well
  - Should be able to improve this, but not an issue for prototype
- Must be at least one non-no-CBs CPU (e.g., CPU 0)
  - Scalability fixes would likely fix this as well.
- No energy-efficiency code: lazy & non-lazy CBs? Non-lazy!
  - But do real-time people even care about energy efficiency?
- No-CBs CPUs' kthreads not subject to priority boosting
  - Rely on configurations restrictions for prototype
- Setting all no-CBs CPUs' kthreads to RT prio w/out pinning them: bad!
  - At least on large systems: configuration restrictions
- Thus, I do not expect no-CBs path to completely replace current CB path

Getting Rid of RCU-bh

- Stated direction from Networking
- Still quite a few uses left: 201 of them!
- But once the uses go, so will the definitions. ;-)
- Which will reduce the size of the -rt patchset

Other RCU Work

Other RCU Work

- Move RCU away from softirq to kthreads (Robustness?)
- Move RCU away from scheduler tick to hrtimer?
- Get rid of TINY_PREEMPT_RCU?
  - Assumes TINY_RCU suffices for memory-constrained systems
- Improved testing and validation (e.g., proof of correctness)
  - Stephen Rothwell's, Dave Jones's, and Wu Fengguang's work very valuable (though sometimes painful – the pain is the value!)
- NUMA? (Sane CPU numbering would help here!)
- Additional use in kernel? (Next slide)
- Use of userspace RCU – userspace is a target-rich environment
- Education/Documentation? (Following slide)

Other RCU Work: Additional Use in Kernel?

Subsystem	Uses	LoC	Uses/KLoC
virt	65	6,400	10.16
ipc	35	8,116	4.31
net	3086	717,501	4.30
security	245	66,990	3.66
kernel	620	187,863	3.30
block	65	28,053	2.32
mm	186	86,486	2.15
lib	66	51,709	1.28
init	2	3,308	0.60
fs	595	1,014,373	0.59
include	266	512,880	0.52
crypto	12	56,913	0.21
drivers	859	8,059,951	0.11
arch	156	2,394,340	0.07
Total	6258	13,194,883	0.47

Summary

- Cache coherence is here to stay, but real-time systems will require software work-arounds for real-fast hardware
- Additional real-time features in flight for RCU
  - Callback offloading, support for Frederic's adaptive ticks, lazy callbacks, remove RCU-bh
- Other RCU work remains to be done, but may be approaching point of diminishing returns in Linux kernel

Summary

- Cache coherence is here to stay, but real-time systems will require software work-arounds for real-fast hardware
- Additional real-time features in flight for RCU
  - Callback offloading, support for Frederic's adaptive ticks, lazy callbacks, remove RCU-bh
- Other RCU work remains to be done, but may be approaching point of diminishing returns in Linux kernel
  - On the other hand I thought I was done with RCU in 1993, 1997, 2004, and 2012, so who knows???

Legal Statement

- This work represents the view of the author and does not necessarily represent the view of IBM.
- IBM and IBM (logo) are trademarks or registered trademarks of International Business Machines Corporation in the United States and/or other countries.
- Linux is a registered trademark of Linus Torvalds.
- Other company, product, and service names may be trademarks or service marks of others.

Questions?